

U.S. Express Mail Label No.: EL 835 825 464 US

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

United States Patent Application for

METHODS AND SYSTEMS FOR SECURING COMPUTER SOFTWARE

Inventor

Maurice Herlihy, a citizen of the United States
residing at 18 Russell Street
Brookline, Massachusetts 02446-2414

METHODS AND SYSTEMS FOR SECURING COMPUTER SOFTWARE

CROSS REFERENCE TO RELATED APPLICATIONS

5 This application claims the benefit of priority of U.S. Provisional applications Ser. Nos. 60/199,934, filed 04/26/2000, entitled "Secure Reactive Software: Managing Fixed-Size Resources"; 60/199,935, filed 04/26/2000, entitled "Secure Reactive Software: Managing Asynchronous Activities"; 60/200,156, filed 04/26/2000, entitled "Secure Reactive Software: Managing Variable-Sizes Resources"; 60/207,560, filed 05/25/2000, entitled "Secure Digital Content Using Leashed Software"; 60/207,559, filed 05/25/2000, entitled "Guaranteeing Fast Access To Leashed Software," the teachings of all of which are incorporated herein by reference.

BACKGROUND OF THE INVENTION

10 The invention pertains to digital data processing and, more particularly, to methods and systems for securing computer software from unauthorized copying, access or use. The invention has application in the sale, licensing and/or leasing of computer programs.

15 Unauthorized software copying or theft was not an issue of great concern to the developers of early computer programs. These were typically leased for use on a single mainframe computer, with pricing based on the number of users (or "seats") entitled to simultaneous access via local or remote terminals. Though software could be copied from computer to computer, programs of value were often so large that surreptitious copying or use was difficult and, typically, relatively easy to detect.

20 With the advent of the personal computer (PC), a different business model emerged. No longer were programs executed on a single computer but, rather, on individual PCs. While some programs are still leased on a per-seat basis, the more common transaction is outright sale with discounts based on numbers of copies sold. This model is flexible enough to accommodate sales to individual sales to private consumers as well as bulk sales to corporations.

Critical to growth of the PC software market is ease of installation. Private consumers and corporate users alike must be able to install software without support from the publisher or technician. Inherent to this, however, is the danger of unauthorized copying. The same technology that works to the benefit of the legitimate software purchaser, notably, "install" disks, network downloads and installation wizards, also works to the benefit of the unauthorized copyist.

While a variety of techniques have been devised to protect against unauthorized copying or use of software, these have often proven too cumbersome for practical use. An object of this invention, accordingly, is to provide improved methods and systems for transforming and executing secured computer software.

A more particular object is to provide such methods and systems as are adapted for use on networked computers and particularly, for example, computers that are "on" the Internet.

Another more particular object is to provide such methods and systems as are adapted for use with business software and game or other entertainment software, alike.

Still another object of the invention is to provide such methods and systems as can be provided at low cost and as consume minimal processing and memory resources.

SUMMARY OF THE INVENTION

The foregoing are among the objects obtained by the invention, which provides improved methods and apparatus for securing computer software against unauthorized use, access, copying and/or functional analysis (e.g., "reverse engineering"). According to one aspect of the invention, such a method involves executing the software so as to make requests that require at least asynchronous responses for continued normal operation. Those responses are generated external to the software and supplied to it, e.g., via a network connection or otherwise. The software continues normal operation as long as it receives the responses within an expected period -- e.g., a period that corresponds to typical latency in responses from the external source -- otherwise, the program ceases normal operation

Further aspects of the invention provide methods as described above in which the process executes on a client device (e.g. a personal computer) and the responses are generated on a server (e.g., operated by the software publisher or at another secured site) which communicates with the client device via a network, such as the Internet. Related aspects provide such methods in which the responses are generated on a coprocessor or other local hardware device that communicates with the protected software via a local bus, for instance.

The invention provides, in still other aspects, methods as described above in which the externally-generated responses are non-deterministic responses and/or otherwise computationally difficult to generate, e.g., without access to source or other programming code underlying the protected software.

Still another aspect of the invention provides methods as described above wherein the protected software performs memory or other resource allocations and wherein continued normal operation depends on at least occasional de-allocations, e.g., to avoid memory or other storage overruns. Such methods include executing requests within the software and utilizing responses to those requests as bases for necessary de-allocations.

Further aspects of the invention provide methods for transforming software to operate as described above and, thereby, to secure it against unauthorized use, access, copying and/or functional analysis..

5 Still further aspects of the invention provide digital data processing systems operating in accord with the above described methods.

Other aspects of the invention provide systems paralleling the operation described above. These and other aspects of the invention are evident in the drawings, description and claims that
10 follow.

11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209

BRIEF DESCRIPTION OF THE DRAWINGS

A more complete understanding of the invention may be attained by reference to the drawings, in which:

5

Figure 1 depicts a transformation according to the invention wherein an original reactive program is transformed into a client program and a server program, each hosted in a client environment and server environment respectively;

10

Figure 2 depicts a transformation according to the invention whereby division of allocation and de-allocation functionality is segregated between the client and server programs;

Figure 3 depicts a stage of the transformation according to the invention whereby over-allocation of dynamic resources is performed;

15

Figure 4 depicts a stage of the transformation according to the invention whereby the de-allocation of dynamic resources is performed;

20

Figure 5 depicts a method of executing protected software according to the invention wherein the random de-allocation of resources occurs during run-time;

Figure 6 depicts a stage of the transformation according to the invention whereby the client program includes steganographic calls to the server.

25

DETAILED DESCRIPTION OF THE ILLUSTRATED EMBODIMENT

While a variety of different techniques exist for protecting software against unlawful copying and distribution, systems which are considered relatively secure include those in which a
5 original program P is split into two programs, a client program C running at a processor controlled by the client, and a server program S running at a processor controlled by the owner and, typically, not readily accessible to the client. The client and server processors operate in communication. If C and S are executed concurrently, together they realize the functionality of the original P. The client cannot execute P by itself, and it is difficult for the client to reconstruct
10 the functionality of P given C and many instances of the communication between C and S, but not S itself. In this way, the owner can use control over C to prevent unauthorized execution of P.

In some embodiments, the owner controlled processor is a secure co-processor or
15 hardware key attached to the client machine with communication occurring over a local bus (see for example US patent 5,754,646 issued to Williams). To save in hardware costs, secure co-processors in the commercial market are usually inexpensive devices with limitations on computing speed and memory size.

20 In other embodiments, the owner-controlled processor is a remote host that communicates with the client host over a network such as the Internet. One system that embodies this approach is described in U.S. Patent No. 6,009,543, entitled "Secure Software System and Related Techniques," the teachings of which are incorporated herein by reference.

25 For many programs, acceptable performance includes the requirement that the program respond to certain inputs within a certain time duration. For brevity, we will call such programs *reactive* programs. Reactive programs include, but are not limited to, programs such as interactive games, word processors, teleconferencing, financial software, database front-ends, players of video or audio, and any other programs that interact with human users by responding
30 to their commands. Reactive programs also include real-time systems such as process controllers one might find in factories, power plants, automobiles, etc.

A major concern with software-splitting techniques is the latency introduced by communication between the client's processor and the owner's processor. It will be appreciated by those of ordinary skill in the art that one cannot easily split a reactive program P into client and server programs C and S in a way that preserves the reaction time of P.

In a coprocessor embodiment, the coprocessor is likely to be substantially slower than the main processor, and the need to buffer data and to share a system bus with other activities (such as memory access) implies that communication delays can be substantial and unpredictable. Moreover, many secure co-processors have limited memory size, which implies that programs and data must be swapped in and out of memory during computation, further increasing communication delays and uncertainty.

In a network embodiment, network delays can be long or unpredictable, and there are many situations in which it is not effective or acceptable to rely on a network to guarantee timely response to inputs.

In either embodiment, if P is split in such a way that C communicates with S in the interval between receiving an input and generating its response, then the observed reaction time of C may be substantially longer than the reaction time of P, and the performance of the split program would be unacceptable to the client.

The illustrated embodiment provides a technique for controlling the use of reactive programs without rendering the reaction time of such programs unacceptable. To this end, it involves splitting a program P so that there is no real-time dependency of the client program C on the server program S.

More particularly, in the discussion that follows, we describe an embodiment in which an original reactive program P (in source, binary, or any intermediate form) is transformed into two programs, C and S, a first (client) storage device having C stored therein, a second (server) storage device having a server program which utilizes S, and execution processors coupled to the

client and server storage devices to execute C and S respectively. With this particular arrangement, a processing system for use with secure reactive software is provided. The system allows the server program to control the execution of the client program C. In one embodiment, the transformation is accomplished by a code transformation processor, a program that receives P and possibly some additional parameters as input, and produces S and C as output. In another embodiment, the transformation is performed directly by a programmer.

Figure 1 depicts a system 10 according to the invention that transforms an original program 101 into a client program 105 and a server program 107, and that executes those programs in view of a set of server tables 108 so as to secure the programs 101, 105, 107 from unauthorized use, access, copying and/or functional analysis (e.g., "reverse engineering").

Illustrated program 101 comprises high level language, object code or other intermediate code, microcode, or other programming instructions to be secured from unauthorized copying, access, use or functional analysis. Though depicted as contained on a CD ROM, it will be appreciated that program 101 can be stored in any known format or on any known medium.

The program 101 is transformed through an automated process (such as by illustrated transformation engine 103) or "by hand" (such as by a computer programmer). The transformation can occur in one or more steps of phases, referred to below as transformation stages one through four that are executed serially (as described) or concurrently with one another. The transformation 103 results in a client program 105, a server program 107 and one or more server tables 108. Those skilled in the art will appreciate that, though the transformation is shown as being effected on an original program 101, in alternate embodiments the client program 105, the server program 107 and server tables 108 can be produced directly (e.g., by the programmer) without need for an original program nor a transformation 103.

Like the original program 101, the client program 105 comprises high level language, object code or other intermediate code, microcode, or other programming instructions. In the illustrated embodiment, the client program 105 is generated in the same form as the original program; however, in other embodiments it can be generated in a different form.

In the illustrated embodiment, the client program 105 is hosted in an environment such as a personal computer 109. In alternate embodiments, it is hosted on any variety of digital data processing devices, from PDAs to video game boards. The client program is transferred to the client device 109 via install disks, downloading, or any other mechanism known in the art for code transfer and installation. Further, when in communication with the server program 107, the client program 105 reacts to inputs in a manner substantially similar as the original program 101 would if hosted in the same environment.

The server program 107 is hosted in a server environment, such as web server 110. However, such hosting can take a variety of well known forms such as taught in U.S. Patent 6,009,543 entitled Secure Software System and Related Techniques by Shavit, or U.S. Patent 5,754,646 entitled Method for Protecting Publicly Distributed Software by Williams et al. As with hosting the client program 105, the server program 107 may be hosted as illustrated on a remote server, or is also suitable for hosting on a secured coprocessor or a client processor with a pre-determined set of secure instructions and memory, or other means similar to the client program 105. The server program is transferred to the server device 110 via install disks, downloading, or any other mechanism known in the art for code transfer and installation.

The server program 107 generates responses to requests from the client program 105, and communicates the responses using a means for communication 112. Further, the server program 107 from time to time randomly initiates responses without requests in a non-deterministic manner. When the server program 107 receives a request, it determines the proper response by using the data stored within the server tables and data structures 108.

The illustrated communication device 112 is the Internet, but it can be appreciated that a variety of communication techniques may be used such as a local bus, wide or local area networks, or a local interface, to name a few.

Many computer programs encompass tasks that are executed as a sequence of steps such that fall into two groups: *active* steps that must be executed immediately to preserve the reactive

nature of the program, and *lazy* steps that may be executed at any point within a given duration without jeopardizing the program's reactive properties. The technique described herein splits such activities of the original program 101 between the client process 105 and server program 107 in the following way. In the client program 105, lazy steps of the original program are replaced by requests to the server. These requests are structured in a way that ensures that an observer inspecting the client program and its executions cannot easily reconstruct the original lazy steps. The server program performs the lazy steps and informs the client program when it does so by asynchronous messages.

A specific example of tasks comprising active and lazy steps is dynamic memory allocation and de-allocation. Figure 2 depicts a transformation of such a task wherein an original program segment 202 is transformed by a transformation stage 204, a part of the transformation 102 (Figure 1), to include requests to the server 208 for data necessary to allocate and de-allocate dynamic memory on device 109. The figure also depicts the generation of the server tables 210 (see, element 108 of Figure 1) during the transformation.

In studying the text that follows, those skilled in the art will appreciate that a block of memory is a contiguous sequence of one or more bytes in a computing device's primary memory. A block b is characterized by two components:

- (1) a starting address $b.addr$, which is the address of the first byte in the block; and
- (2) a size $b.size$, which is the number of bytes in the block.

A block b is empty if $b.size$ is 0. A byte of memory x is in a block b if the address of x is greater than or equal to $b.addr$ and less than $b.addr + b.size$. A block c is contained within block b if every byte in c is also in b . A block b can be split into two smaller blocks c and d , where $b.addr = c.addr$, $d.addr = c.addr + c.size$, and $d.size = b.size - c.size$. Similarly, c and d can be merged to form b .

A computer program creates and disposes of data structures within memory blocks as it executes. To support such activity, the program maintains a free-pool of unused memory. To create a data structure of particular size, the program allocates a block of memory large enough to hold the data structure, thereby removing that memory from the free-pool. When the program

no longer requires that data structure, it returns the memory block to the free-pool, thus making the memory available for other purposes. Typically, run-time management libraries are used to allocate and de-allocate memory blocks. For example, in the C-language the statement:

5 *obj_ptr = malloc(obj_size);*

allocates a block of *obj_size* bytes, returning the starting address of the block in *obj_ptr*. Further, the statement:

free(obj_ptr);

10

returns that block of memory to the free-pool. It will be appreciated by those of ordinary skill in the art that other techniques of memory management can easily be translated to use equivalent methods for the allocation and de-allocation of memory blocks or segments.

15

Referring to Figure 2, the transformation stage 204 translates the *malloc* instructions 212, 214 and *free* instructions 216, 218 instructions of an original program 202 such that the dynamic allocation/de-allocation instructions are divided between the activities of the client 206 and the server 208 in such way that as long as the client and server remain in communication, the client will allocate and free memory correctly.

20

For example, the code segment listing that follows corresponds to the original reactive program segment 202 in Figure 2. The number at the beginning of each line in the listing represents the program counter or other index and the text represents High-Level language:

55: x=5;	60: y=x;
56: malloc(y);	61: x=z;
57: y = 6+x;	62: free(z);
58: malloc(z);	63: x=2;
59: z=y;	64: free (y);

25

After the illustrated transformation stage depicted in Figure 2, the resulting process segment 206 would be:

55: x=5;	62: y=x;
56: malloc(y);	63: send(m);
57: send(m);	64: x=z;
58: y=6+x;	65: send(m);
59: malloc(z);	66: x=2;
60: send(m);	67: send(m);
61: z=y;	

and the server tables 210 would contain:

57	malloc(y)
60	malloc(z)
63	dummy
65	free(z)
67	free(y)

In this example, from beginning to end, the *free* instruction 216 in the original program segment 202 is transformed into the *send* instruction 228 in the client segment 206 and the *free* table entry 232 in the server table, *free_table* 208. The look-up index of 64 corresponds to the program counter in the process segment 206 at which the *send(m)* message request 228 is executed. While the illustrated embodiment references use the program counter as the look-up index into the server table, one skilled in the art can recognize that any random sequence of unique identifiers is applicable to the transformation. Further, it can be noted that the *malloc* operations 212, 214 in the original program segment 202 are present in the process segment 220, 224. It is not obvious which of the *send* instructions 228, 234, 230 correspond to which *free* instruction 216, 218. Furthermore the responses sent from the server program 208 to the client 206 need not be in the same order as the requests from the client 206 to the server program. More specifically, a *free* corresponding to a given *malloc* operation cannot be determined without the server table 210. Without knowing where in the code the *free* messages occur, generating the functionality without analysis of the server table is difficult and could result in either running out of memory or freeing variables that are still in use. The problem of adding new *free* instructions without knowing the tables can be shown to be NP-Hard.

The transformation also has the capability of over-allocating dynamic resources, and randomly de-allocating the over-allocated portion during run-time such that it is computationally hard to learn the appropriate responses from the communication history. Consider the following operations:

s.remove(b) removes from *s* all blocks contained in *b*;
s.add(b) adds *b* to *s*; and
s.choose removes and returns an arbitrary block from *s*.

It will be appreciated by one skilled in the art that a set of blocks can be implemented in a variety of ways such as arrays, trees and linked lists to name a few. After transformation by the invention, information about blocks of memory in use is split between the client and the server as follows: the client keeps track of a set of blocks, *client_set* and the server keeps track of a set of blocks, *server_set*. Each block in *server_set* is a sub-block in *client_set* that is not actually used by the client program. One way in which this is accomplished is by over-allocating resources used by the client program. Whether a memory byte is in use by the client can be determined by examining both *client_set* and *server_set*, but it is computationally hard to determine from the *client_set* alone.

To illustrate this method, consider the following statement at line 82 of the original program segment in Figure 3:

obj_ptr = malloc(obj_size); 302

where *obj_size* is a variable containing the size of the object, and *obj_ptr* is assigned a pointer to the beginning of the object. After the transformation, the client contains:

obj_ptr = malloc(obj_over_size); 304
 .
 .
 .
send(m); 306

where,

$$obj_over_size \geq obj_size;$$

and m is a message containing at least the current value of C 's program counter and the values of some or all of its local variables. This transformation causes C to allocate a block of memory at least large enough to hold the object and then to send a message containing at least its program counter and local variables to S . Note that the *send* instruction 306 at line 110 need not appear next to the *malloc* instruction 304 at line 85, but may be separated by some arbitrary or random number of instructions, including other send instructions.

The server program S is initialized with table *malloc_table* 310 that identifies the program counter (pc) values in C at which memory blocks are allocated. Each time S receives a message containing the current program counter and local variables of C , it looks up pc in *malloc_table*. If the statement at location pc is a call to *malloc*, then S reconstructs from the local variables the address of the newly-allocated block (obj_ptr), the size of the block (obj_over_size), and the portion of the block actually in use (obj_size), and adds the block with address $obj_ptr + obj_size$ and size $obj_over_size - obj_size$ to the set *server_set*. This program is illustrated as follows 318:

```

while (true) {
    m = receive();
    pc = m.pc;
    if (malloc_table.lookup(pc)) {
        b = new block(m.obj_ptr + m.obj_size,
                     m.obj_over_size - m.obj_size);
        server_set.add(b);
    }
}

```

Another stage of the transformation depicted in Figure 4 shows the transformation of *free* instructions. The *free(obj_ptr)* 402 instruction in the original program segment at line 125 is transformed to into a *send(m)* 404 instruction in the client program as shown at line 150, and an entry is placed in the server table, *free_table* 406 at position 150 corresponding to the program counter in the client program. When the *send(m)* instruction 404 is executed from the client

location 150, where m contains the program counter and some or all of its local variables, the server performs a look-up in the *free_table* 406 to determine the proper action to take. If the statement at location pc is a call to *free*, then S reconstructs from the local variables the address of the block b to be freed. It then adds to *server_set* every block contained in b . In this way, it is difficult to determine when a *free* is actually performed without access to the server table. It is not shown in the figure but should be obvious to one skilled in the art that the responses to free instructions need not be in the same order as the requests are received.

The program segment within the server program to implement after this stage of the transformation could be as follows:

```

while (true) {
    m = receive ();
    pc = m.pc;
    if (malloc_table.lookup(pc)) {
        b = new block(m.obj_ptr + m.obj_size,
                     m.obj_oversize - m.obj_size);
        server_set.add(b);
    } else if (free_table.lookup(pc)) {
        b = new block(m.obj_ptr + m.obj_size,
                     m.obj_oversize - m.obj_size);
        server_set.add(b);
    }
}

```

A further stage of the transformation depicted in Figure 5 allows the server program to periodically remove an arbitrary block of memory during run-time, that is allocated but not actually used by the client program. For example, consider the transformation stages described above using a memory block b 316 (Figure 3). The *server_set* 504 represents block a which is the over-allocation portion of block b as described in this illustrated embodiment and above. The server S removes block a from the *server_set* and splits a arbitrarily into three blocks a_0 , a_1 and a_2 where a_0 or a_2 or both may be empty such that if a_0 is not empty it is placed back into the *server_set* and if a_2 is not empty it is placed also placed back into the *server_set*. Then S sends a message to C that:

$m.addr = a_1.addr;$
 $m.size = a_1.size;$

When the client receives the message, it removes from *client_set* 514 the sub-block b_1 containing $m.addr$ as follows: the client program splits block b 518 into b_0 , b_1 and b_2 where 516:

$b_0.addr = b.addr;$
 $b_0.size = m.addr - b.addr;$
 $b_1.addr = m.addr$
 $b_1.size = m.size;$
 $b_2.addr = m.addr + m.size;$ and
 $b_2.size = b.size - b_0.size - b_1.size.$

Further, if b_0 is not empty, the client places b_0 back into *client_set* 514. Also, if b_2 is not empty, the client places b_2 back into *client_set* 514. This transformation permits the server to return to the client blocks of memory that were allocated but not actually used. Note that these blocks could be the result of either over-allocations or freed memory which the server knows about via old *free* messages it received.

In still another stage of the transformation as depicted in Figure 6, instructions are placed within the client program 602 comprising:

$send(m);$ 604

instructions, where m is a message containing the current program counter or other index and the value of some or all of the client's local variables. The server 606 maintains a server table

dummy_table 608 where the server takes no action if the table entry corresponding to the program counter is a *dummy* operator. In the illustrated embodiment, these statements are executed frequently enough that analysis of the client would not distinguish among the message transmission statements introduced in the transformation stages as discussed above, and further, analysis of the message traffic between the client and server cannot easily track which subset of the memory in *client_set* is actually in use. Thus, the message transmission statements introduced in this transformation provide steganographic protection for the message transmission statements introduced in the earlier transformation.

It is appreciated that in some programs there may be a substantial delay between the time at which the program allocates a memory block, and the time that block is first used. Such activity is a lazy allocation, and provides an alternative transformation stage appropriate for lazy allocations. Consider a program P containing a first statement in the form:

5 *obj_ptr = malloc(obj_size);*

and a second statement in the form:

initialize(obj_ptr);

10

which initializes the contents of the block *b* such that

b.addr = obj_ptr.

15

The first and second statements may be at different positions in P, and there may be a delay between their executions. An embodiment of the invention provides a sequence of transformation stages for such programs.

20

First, a stage of the transformation may apply the following transformation to the first statement. The client will allocate a memory block large enough to hold a pointer, initialize that block to hold a special value, and send the current program counter and local variables *obj_size*, the object size, and *future_ptr*, the address of the newly-allocated block to the server.

25 *future_ptr = malloc(4);*
 **future_ptr = null;*

Here, it is assumed that four bytes are large enough to hold a pointer, and *m* is a message containing the current value of client C's program counter and the value of some or all of C's local variables. Notice that after the transformation, the client cannot easily deduce the size of the object from the transformed code.

30

The server program S is initialized with a table *lazy_malloc_table* similar to the server tables as described above that identifies the program counter values in C at which lazy allocations occur. Each time S receives a message containing the current program counter value

35

pc and local variables of *C*, it looks up *pc* in the *lazy_malloc_table*. If the statement at location *pc* is a lazy allocation, then *S* reconstructs from the local variables the values of *obj_size* and *future_ptr*. The server *S* then removes from *server_set* a block *b* of size greater than or equal to *obj_size*. The server program *S* splits *b* into three blocks, *b*₀, *b*₁, and *b*₂, where *b*₁.size =

5 *m.obj_size*.

If *b*₀ is not empty, *S* places *b*₀ back into *server_set*. If *b*₂ is not empty, *S* places *b*₂ back into *server_set*. The server then sends *future_ptr* and *b*₁.addr to the client.

10 When the client receives *b*₁.addr from the server, it stores that value in the block whose address is *future_ptr*.

**future_ptr = b₁.addr;*

15 The Client's second statement is transformed into two statements: a loop that waits for *future_ptr* to be initialized by the Server's message, and the initialization of the block;

```

while (*future_ptr == null) {
  obj_ptr = *future_ptr
  free(future_ptr);
  initialize(obj_ptr);
}

```

20 In a preferred embodiment, this transformation would be applied to statements such that

25 the delay between executing the first and second statements exceeds the round-trip communication time between the client and the server. In this situation, *C* will not need to execute the loop statement more than once.

30 The above discussion has illustrated an embodiment using variable size resources, but programs often manage pools of fixed-size resources. Such resources include but are not limited to disk pages, memory pages, file descriptors, and fixed-size data structures. For brevity, we disclose the invention in terms of disk pages, but it will be appreciated by those of ordinary skill in the art that these techniques can be applied to any fixed-size resource.

A *disk page* is a contiguous sequence of one or more bytes on a magnetic disk. A page p is characterized by a starting address $p.addr$, which identifies the page's location on the disk. All disk pages have the same size, denoted here by P . A pool of pages is a data structure that keeps track of a plurality of pages. For each page, the pool determines whether the page is in use (allocated) or not in use (*free*). A pool provides the following operations. The call

$page_addr = pool.allocate();$

allocates a page, returning the newly-allocated page's address. The call

$pool.free(page_addr);$

where $page_addr$ is the address of a page previously allocated by `allocate`, returns that page to the pool. The call:

$pool.mark(page_addr);$

marks a specific free page as allocated. The call

$page_addr = pool.choose();$

returns the address of an arbitrary allocated page (or a distinguished value null if none exists).

Run-time libraries typically provide a variety of more specialized disk page allocation calls, or other calls of equivalent functionality. It will be appreciated by those of ordinary skill in the art that programs that manage disk pages using other techniques can easily be re-written to use a run-time library of equivalent functionality.

The transformation of program P is accomplished as described below. This non-limiting example provides a transformation stage that divides the management of disk pages (or any other fixed-size resource) by P between C and S in such a way that as long as C and S remain in communication, C will allocate and free disk pages correctly. Moreover, C will respond to inputs within the same required duration as P .

In this described embodiment, information about which disk pages are in use is split between the client program C and the server program S as follows. The client program C keeps

track of a pool of pages *client_pool*. The server keeps track of a pool of pages *server_pool*. Both *client_pool* and *server_pool* manage the same set of pages. In the preferred embodiment, each page in *client_pool* is a page allocated by C, and each page in *server_pool* is a page allocated by C but not actually in use by C. Whether a page is in use by C can thus be determined by
 5 examining both *client_pool* and *server_pool*, but cannot necessarily be ascertained from *client_pool* alone.

For example, consider the following statement of P:

10 *page_addr* = *allocate*();

the statement is transformed into the following two statements in C:

15 *page_addr* = *allocate*();
send_server(*m*);

Here, *m* is a message containing the current value of C's program counter, and the value of some or all of C's local variables. This transformation causes C to allocate a disk page and then to send a message containing its program counter and local variables to S. These statements
 20 may be executed one right after the other, or they may be separated by other statements.

In a further stage of the transformation, the additional allocation requests to C, could take the form:

25 *page_addr* = *allocate*();
send_server(*m*);

In a preferred embodiment, these additional allocation requests make it difficult for the client to determine which allocations correspond to allocations in P, and which are introduced by
 30 the transformation. As in the first transformation, these statements can be executed one after the other, or they may be separated by other statements. For brevity, we refer to the allocation requests introduced in this transformation as *spurious* allocations.

The server program S is initialized with a table *alloc_table* that identifies the program
 35 counter values in C at which spurious allocations occur. Each time S receives a message containing the current program counter value *pc* and local variables of C, it looks up *pc* in

alloc_table. If the statement at location *pc* is a spurious allocation, then S reconstructs from the local variables the address of the newly-allocated disk page (*page_addr*), and marks the disk page address *page_addr*:

5 *server_pool.mark(page_addr);*

This program is illustrated as follows:

```

10           while (true) {
              m = receive();
              pc = m.pc;
              if (alloc_table.lookup(pc))
                 server_pool.mark(m.page_addr);
              }

```

A further stage of the transformation transforms statements in which P frees a disk page previously allocated by allocate:

free(page_addr);

In the client program C, this statement is transformed into a message transmission:

send_server(m);

where *m* is a message containing the current value of C's program counter and the value of some or all of C's local variables.

The server program S maintains a table *free_table* of the program counter values in C at which a disk page is freed. Each time S receives a message containing the current program counter value *pc* and local variables of C, it looks up *pc* in *free_table*. If the statement at location *pc* is a call to *free*, then S reconstructs from the local variables the address of the page *p* to be freed. It then marks that page as in *server_pool*. The resulting server program is shown below:

```

35           while (true) {
              m = receive();
              pc = m.pc;
              if (alloc_table.lookup(pc))
                 server_pool.mark(m.page_addr);
              else if (free_table.lookup(pc))
40               server_pool.mark(m.page_addr)

```

}

In a fourth stage of the transformation, the server program S periodically performs the following actions.

1. It removes one or more disk pages from *server_pool*;
2. It creates a message *m* whose fields include the address of each disk page removed in Step 1; and
3. It sends *m* to C.

This transformation permits S to return to C disk pages that were spuriously allocated by C or previously freed by C.

In a fifth stage of the transformation, message transmission statements are added to C. Each message transmission has the form:

send_server(m);

where *m* is a message containing the current value of C's program counter, and the value of some or all of C's local variables.

In a preferred embodiment, these statements are executed frequently enough that the client cannot distinguish between them and the message transmission statements introduced in the prior transformation stages.

A client monitoring the message traffic between C and S thus cannot easily track which disk pages in *client_pool* are actually in use, because real free messages cannot be distinguished from fake ones. The message transmission statements introduced in this transformation thus provide steganographic protection for the message transmission statements introduced in the earlier transformations.

In a preferred embodiment, the activities of the client program C are never delayed by waiting for a message from S, so the time needed for the transformed programs C and S to respond to inputs will not be substantially longer than the time needed for the original program P to respond. Because the client cannot determine, by inspecting C, when disk pages are freed, C will eventually run out of disk pages if it is executed without communicating with S.

It is obvious to one skilled in the art that a lazy allocation scheme can be devised for fixed size resources in a manner similar in nature to that of variable sized dynamic memory allocation.

5

Finally, the server tables must be secured against unauthorized access. It is undesirable to require every server to maintain a long-lived database of *malloc*, *free* and other tables for each client. Therefore, a method of co-located client-server programs is described herein as follows that is applicable for distribution of both the client program and server tables (and possibly parts of the server program) to the client site on CD, DVD or other computer readable media, for example. The security of the transformation relies on ensuring that an unauthorized user never obtains access to the server tables. One can achieve this goal by keeping the tables encrypted where the encryption key is known only to authorized servers. The vendor splits the original program into a process and server with an encrypted set of server tables, where the encryption key is known only to the vendor. In order to execute the client, it sends the encrypted tables to a server, where they are decrypted and used by the server until such time as the client completes, when the tables are deleted from the server. Therefore, the server does not need to keep a permanent database of server tables, and yet the scheme is secure because the client never observes the unencrypted server tables.

10

15

20

Described above are methods and systems meeting the desired objects. It will be appreciated that the illustrated embodiment is merely an example of the invention and that other embodiments, incorporating modifications thereto fall within the scope of the invention. Thus, by way of non-limiting example, it will be appreciated that the transformation 103 can be performed with fewer or more transformation stages than those discussed above, and can be performed by a programmer or software engine. Moreover, it may perform those stages serially or concurrently. The transformed resources managed need not be linked to storage resources, but may also be sub-processes that are created ("allocated") and eliminated ("freed"). Further, it will be appreciated that though the examples are illustrated using the C programming language, the method is applicable for other high-level languages, object, assembly, microcode and any other intermediate instruction set. Still further, it will be appreciated that the mechanisms described

25

30

above can be used, not only to secure the client program from unauthorized use, access, copying and/or functional analysis, but also to permit control of the client from the server.

In view of the foregoing, what we claim is:

5

10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
105
110
115
120
125
130
135
140
145
150
155
160
165
170
175
180
185
190
195
200
205
210
215
220
225
230
235
240
245
250
255
260
265
270
275
280
285
290
295
300
305
310
315
320
325
330
335
340
345
350
355
360
365
370
375
380
385
390
395
400
405
410
415
420
425
430
435
440
445
450
455
460
465
470
475
480
485
490
495
500
505
510
515
520
525
530
535
540
545
550
555
560
565
570
575
580
585
590
595
600
605
610
615
620
625
630
635
640
645
650
655
660
665
670
675
680
685
690
695
700
705
710
715
720
725
730
735
740
745
750
755
760
765
770
775
780
785
790
795
800
805
810
815
820
825
830
835
840
845
850
855
860
865
870
875
880
885
890
895
900
905
910
915
920
925
930
935
940
945
950
955
960
965
970
975
980
985
990
995